Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Information Processing Letters 86 (2003) 299–302

Information Processing Letters

www.elsevier.com/locate/ipl

From a simple elimination ordering to a strong elimination ordering in linear time

J. Sawada [a,*,1], J.P. Spinrad [b,2]

[a] *Department of Computer Science, University of Toronto, Canada*
[b] *Department of Computer Science, Vanderbilt University, Nashville, TN 37235, USA*

Received 25 September 2002; received in revised form 13 January 2003

Communicated by K. Iwama

Abstract

We present a linear time algorithm for transforming a simple elimination ordering of a strongly chordal graph into a strong elimination ordering.

Keywords: Graph algorithms; Strongly chordal graph; Simple elimination ordering; Strong elimination ordering; Graph recognition

1. Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E where $|V| = n$ and $|E| = m$. A graph is *strongly chordal* if it is chordal (every cycle of length 4 or more contains a chord) and if every even cycle of length 6 or more contains a chord splitting the cycle into two odd length paths. Let $N(v)$ denote the neighborhood of a vertex v and let $N[v]$ denote the closed neighborhood $N(v) \cup \{v\}$. For $A \subseteq V$, we let $G(A)$ denote the subgraph of G induced by A. A vertex v is said to be *simple* if the set $\{N[u]\colon u \in N(v)\}$ can be linearly ordered by set inclusion. Alternatively, a vertex v is simple if for any two vertices $x, y \in N(v)$ either $N[x] \subseteq N[y]$ or $N[y] \subseteq N[x]$. An ordering $v_1, v_2, \ldots, v_n$ is called a *simple elimination ordering* if for each $1 \leqslant t \leqslant n$, the vertex v_t is simple in $G(\{v_t, \ldots, v_n\})$.

Theorem 1 [2]. *A graph G is strongly chordal if and only if it has a simple elimination ordering.*

Simple elimination orderings are not the only vertex orderings that characterize strongly chordal graphs, as we see in the next theorem. An ordering $v_1, \ldots, v_n$ is called a *strong elimination ordering* if it is a simple elimination ordering and for each $i < j < k$ where $v_j, v_k \in N(v_i)$ we have $N[v_j] \subseteq N[v_k]$ with respect to $G(\{v_i, \ldots, v_n\})$. From this definition we see that a strong elimination ordering is a simple elimination ordering, but a simple elimination ordering is not necessarily a strong one.

Theorem 2 [2]. *A graph G is strongly chordal if and only if it has a strong elimination ordering.*

* Corresponding author.
E-mail addresses: jsawada@cs.toronto.edu (J. Sawada), spin@vuse.vanderbilt.edu (J.P. Spinrad).

[1] Research supported by NSERC.
[2] Supported by National Science Foundation Grant 9820840.

0020-0190/03/$ – see front matter
doi:10.1016/S0020-0190(03)00228-X

Currently, the fastest algorithms for recognizing strongly chordal graphs run in time $O(m \log n)$ due to Paige and Tarjan [5] and $O(n^2)$ for dense graphs due to Spinrad [6]; Uehara has retracted his claim of a linear time algorithm to solve the problem, citing fundamental flaws in the algorithm of [7]. Since many closely related classes of graphs like chordal graphs, interval graphs and chordal comparability graphs can be recognized in linear time, it seems plausible that strongly chordal graphs can be as well. Thus, one of the most interesting open problems regarding strongly chordal graphs is whether or not we can recognize them in linear time.

Strong elimination orderings also have an interesting matrix interpretation. The *augmented adjacency matrix* of G is the matrix obtained from the adjacency matrix by substituting 1 entries for 0s on the main diagonal. A vertex ordering is a strong elimination ordering if and only if using this ordering for the rows and columns of the augmented adjacency matrix, the resulting matrix contains no induced $\binom{11}{10}$ (also called a Γ). Lubiw [4] was able to show that it is possible to determine whether a matrix is Γ-free in linear time. Therefore, any algorithm for finding strong orderings in linear time would immediately give a linear algorithm for recognizing strongly chordal graphs.

Since every strong elimination ordering is simple but not vice-versa, a potentially easier open problem is to find a simple elimination ordering in linear time. However, given a simple elimination ordering, it was previously not known whether we could then recognize strongly chordal graphs in linear time. This paper addresses this issue by developing a linear time algorithm for transforming a simple elimination ordering into a strong elimination ordering. Thus, if a linear time algorithm for finding a simple elimination ordering of a strongly chordal graph is discovered, then we will also be able to recognize strongly chordal graphs in linear time.

For a special class of strongly chordal graphs, the chordal comparability graphs, a linear time simple elimination scheme was developed by Borie and Spinrad [1].

2. Algorithm

In this section we present an algorithm for transforming a simple elimination ordering into a strong elimination ordering. We start with an example of a simple elimination ordering that is *not* a strong elimination ordering. Observe that the ordering a, b, c, d, e, f, g is a simple elimination ordering for the graph shown in Fig. 1. Now consider the vertex b in the ordering. It has neighbors e and f such that $e < f$ in the ordering. But since the vertex c is a neighbor of e and not f, and since $b < c$, the ordering is not a strong ordering. In this example we see that the problem is the relative ordering of the higher neighbors (e and f) of the vertex b.

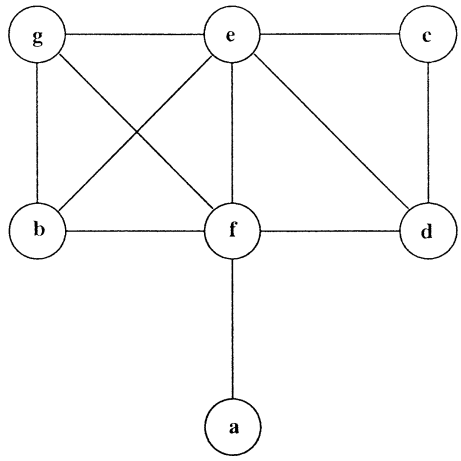

Fig. 1. A strongly chordal graph.

Assume that we have a simple elimination ordering $v_1, v_2, \ldots, v_n$ of a strongly chordal graph G. The following function partitions the vertices into a list $\mathcal{L}$ of disjoint sets.

Function MakeSets
(G: strongly chordal graph,
$[v_1, \ldots, v_n]$: simple elimination ordering)
returns ordered list of sets that partition V;
(1) $\mathcal{L} :=$ empty list of sets;
(2) **for** $t := 1$ **to** n **do begin**
(3) **if** $v_t \in V$ **then begin**
(4) $S := \{u \mid u \in N[v_t]$ **and** $\deg(u) = \deg(v_t)\}$;
(5) append S to $\mathcal{L}$;
(6) remove S from V, updating
the neighborhoods and degrees;
(7) **end**;
(8) **end**;
(9) return ($\mathcal{L}$);

To illustrate, the function MakeSets($G, [a, b, c, d, e, f, g]$), where G is the graph from Fig. 1, returns the list of sets $\mathcal{L} = \{a\}, \{b, g\}, \{c\}, \{d, e, f\}$. Now by visiting each set in order and arbitrarily outputting the vertices within each set, we can obtain a new ordering a, b, g, c, d, e, f. To simplify the discussion, we will let $\langle\mathcal{L}\rangle$ denote an arbitrary ordering obtained from $\mathcal{L}$ in this fashion. In the following lemma we prove that an ordering $\langle\mathcal{L}\rangle$ is also a simple elimination ordering.

Lemma 1. *If $\mathcal{L} = S_1, S_2, \ldots, S_r$ is returned from the function* MakeSets($G, [v_1, \ldots, v_n]$) *then $\langle\mathcal{L}\rangle = u_1, \ldots, u_n$ is a simple elimination ordering. Moreover, if there exists $i < j < k$ such that (u_i, u_j), (u_j, u_k), and (u_i, u_k) are edges in G and there exists $i < l$ such that (u_j, u_l) is an edge in G, but (u_k, u_l) is not, then u_j and u_k must belong to the same set in $\mathcal{L}$.*

Proof. Clearly the set S_1 contains v_1, which is a simple vertex in G. Any other vertex $x \in S_1$ must be a neighbor of v_1 and have the same degree. Now suppose that $N[x] \neq N[v_1]$. This implies that there exists some vertex y adjacent to v_1 that is not adjacent to x. This contradicts the fact that v_1 is a simple vertex in G. Thus $N[x] = N[v_1]$ which implies that x (and every vertex in S_1) is simple in G. Using this same argument, it follows that each vertex in S_i is simple in the graph $G(V - S_1 - S_2 - \cdots - S_{i-1})$. Thus $\langle\mathcal{L}\rangle$ is a simple elimination ordering.

Now suppose there exists $i < j < k$ such that (u_i, u_j), (u_j, u_k), and (u_i, u_k) are edges in G and there exists $i < l$ such that (u_j, u_l) is an edge in G, but (u_k, u_l) is not. If u_j and u_k are in different sets in the list $\mathcal{L}$, then since $j < k$, u_j must appear in a set that precedes the set containing u_k in $\mathcal{L}$. Since u_j is simple in $G(\{u_j, \ldots, u_n\})$ it must be the case that $l < j$. This implies that u_k must be adjacent to some vertex z that is not adjacent to u_j in $G(\{u_j, \ldots, u_n\})$; otherwise u_k would be in the same set as u_j. This however contradicts the fact that u_i is simple in $G(\{u_i, \ldots, u_n\})$. Thus u_j and u_k must belong to the same set in $\mathcal{L}$. □

Recall that we can obtain a new ordering a, b, g, c, d, e, f from the list of sets returned by MakeSets($G, [a, b, c, d, e, f, g]$) for the graph in Fig. 1. We have proved that this is a simple elimination ordering, but notice that it is *not* a strong elimination ordering. There is, however, an ordering of the vertices within each set of $\mathcal{L}$ that will result in a strong elimination ordering, namely a, b, g, c, f, d, e. In fact, it turns out that there will always be such an ordering of the vertices within each set that will lead to a strong elimination ordering. We verify this claim by construction.

Function SimpleToStrong
(G: strongly chordal graph,
$[v_1, \ldots, v_n]$: simple elimination ordering);
returns strong elimination ordering;
(10) $\mathcal{L} := \mathcal{L}' :=$ MakeSets($G, [v_1, \ldots, v_n]$);
(11) $v_1, \ldots, v_n := \langle\mathcal{L}\rangle$;
(12) **for** $t := n$ **down to** 1 **do begin**
(13) **for** each set $S \in \mathcal{L}$ containing a vertex in $N[v_t]$ **do begin**
(14) **if** $S - N[v_t] \neq \emptyset$ **then**
(15) replace S in the list $\mathcal{L}$ with the two sets $S - N[v_t]$, $S \cap N[v_t]$;
(16) **end**;
(17) **end**;
(18) return ($\langle\mathcal{L}\rangle$);

This function starts by obtaining a list of sets $\mathcal{L}$ returned by MakeSets($G, [v_1, \ldots, v_n]$). A copy of this list $\mathcal{L}'$ is made for discussion purposes only. In line (11) the simple elimination ordering $v_1, \ldots, v_n$ is updated to one that can be obtained from $\mathcal{L}$. Then in lines (12)–(17) each set is partitioned into a list of sets by visiting each vertex in reverse order of the new simple elimination ordering. As a vertex v_t is visited, we replace each set $S \in \mathcal{L}$ that contains a neighbor of v_t with two sets (as long as $S - N[v_t]$ is nonempty) $S - N[v_t]$ and $S \cap N[v_t]$. It is important that the set $S - N[v_t]$ is placed before $S \cap N[v_t]$ in the list $\mathcal{L}$. This process effectively places a partial order on the vertices of each set S in the original list $\mathcal{L}'$. Thus, the ordering $\langle\mathcal{L}\rangle$ we return in line (18) is one that can also be obtained from $\mathcal{L}'$ and hence it is a simple elimination ordering. As an example, the function SimpleToStrong($G, [a, b, c, d, e, f, g]$), where G is the graph from Fig. 1, returns $\langle\mathcal{L}\rangle$ where $\mathcal{L} = \{a\}, \{b, g\}, \{c\}, \{f\}, \{d\}, \{e\}$.

Theorem 3. *If $v_1, \ldots, v_n$ is a simple elimination ordering of a strongly chordal graph G, then the function* SimpleToStrong($G, [v_1, \ldots, v_n]$) *outputs a strong elimination ordering of G.*

Proof. Let $u_1, u_2, \ldots, u_n$ be the ordering returned in line (18). This is a simple elimination ordering that can also be obtained from the list $\mathcal{L}'$. Now suppose there exists $i < j < k$ such that (u_i, u_j), (u_i, u_k), and (u_j, u_k) are edges in G and there exists $i < l$ such that (u_j, u_l) is an edge in G, but (u_k, u_l) is not. From Lemma 1, u_j and u_k must belong to the same set in $\mathcal{L}'$. Also, since u_i is simple in $G' = G(\{u_i, \ldots, u_n\})$ we must have $N[u_k] \subset N[u_j]$ with respect to G'. Thus, after the iteration of the **for** loop in (12) when $t = l$, the vertex u_k will be in a set that precedes the set containing u_j. This contradicts $j < k$. Thus, by definition, $u_1, \ldots, u_n$ is a strong elimination ordering. □

3. Analysis

In the previous section we have outlined an algorithm for transforming a simple elimination ordering into a strong elimination ordering via the function SimpleToStrong$(G, [v_1, \ldots, v_n])$. We will show in this section that the algorithm can be implemented to run in linear time.

First we must analyze the function MakeSets$(G, [v_1, \ldots, v_n])$. Notice that when we create each set S, we need only consider the neighbors of the current vertex v_i. Thus, to create all sets $S \in \mathcal{L}$ we consider at most a total of m vertices. As we remove each vertex from a set S from V, we must update the degrees and neighborhoods of each vertex. Again this work is proportional to the number of edges in the graph. Thus MakeSets$(G, [v_1, \ldots, v_n])$ runs in time $O(m)$.

We now examine the code in lines (12)–(17). Each vertex belongs to exactly one set S. By maintaining an appropriate data structure, we can obtain the set a given vertex belongs to in constant time. Thus, we can obtain the sets containing the vertices in $N[v_i]$ and create the new sets $S \cap N[v_i]$ and $S - N[v_i]$ in time $O(N[v_i])$. This procedure is equivalent to the step of subdividing sets into neighbors and non-neighbors of v as in chordal graph recognition; details of the chordal graph implementation are given in [3]. Thus the total amount of work done in this step is $O(m)$.

Theorem 4. *The functions* SimpleToStrong$(G, [v_1, \ldots, v_n])$ *and* MakeSets$(G, [v_1, \ldots, v_n])$ *can be implemented to run in* $O(m)$ *time.*

4. Future work

In this paper we have presented an algorithm for transforming a simple elimination ordering into a strong elimination ordering. Using this result, if we can find a simple elimination ordering in linear time, then we can find a strong elimination ordering in linear time, and thus we can recognize strongly chordal graphs in linear time. Thus, a critical open problem is to discover a linear time algorithm for finding a simple elimination ordering of a strongly chordal graph.

References

[1] R.B. Borie, J.P. Spinrad, Construction of a simple elimination scheme for a chordal comparability graph in linear time, Discrete Appl. Math. 91 (1999) 287–292.
[2] M. Farber, Characterizations of strongly chordal graphs, Discrete Math. 43 (1983) 173–189.
[3] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[4] A. Lubiw, Doubly lexical orderings of matrices, SIAM J. Comput. 16 (1987) 854–879.
[5] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (1987) 973–989.
[6] J.P. Spinrad, Doubly lexical ordering of dense 0–1 matrices, Inform. Process. Lett. 45 (1993) 229–235.
[7] R. Uehara, Linear time algorithms on chordal bipartite and strongly chordal graphs, in: ICALP, 2002, pp. 993–1004.